

The `goplumb` package for manipulating `plumb` messages

(version 0.4.0)

Alexander Sychev ([santuccio@gmail.com](mailto:santuccio@gmail.com))

**1. Introduction.** In a great operating system **Plan 9** there is a **plumber** - a filesystem for interprocess messaging. The **goplumb** package is implemented to manipulate such messages. The main target of the package is support of **plumber** from **Plan 9 from User Space** project [http:// swtch.com/plan9port/](http://swtch.com/plan9port/).

**2. Implementation.**

```

// This file is part of ahist
//
// Copyright (c) 2013, 2020 Alexander Sychev. All rights reserved.
//
// Redistribution and use in source and binary forms, with or without
// modification, are permitted provided that the following conditions are
// met:
//
// * Redistributions of source code must retain the above copyright
// notice, this list of conditions and the following disclaimer.
// * Redistributions in binary form must reproduce the above
// copyright notice, this list of conditions and the following disclaimer
// in the documentation and/or other materials provided with the
// distribution.
// * The name of author may not be used to endorse or promote products derived from
// this software without specific prior written permission.
//
// THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
// "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
// LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
// A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
// OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
// SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
// LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
// DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
// THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
// (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
// OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
// Package goplumb provides interface to plumber - interprocess messaging from Plan 9.
package goplumb

import(
    <Imports 6>
)

type(
    <Types 4>
)

var(
    <Variables 9>
)

```

3. Let's describe a begin of a test for the package. The `plumber` will be started for the test.

```

<goplumb_test.go 3> ≡
package goplumb
import(
    "os/exec"
    "testing"
    "bytes"
    "time"
    "9fans.net/go/plan9"
    "9fans.net/go/plan9/client"
    <Test specific imports 22>
)
const rule = `type_is_text\nsrc_is_Test\nplumb_to_goplumb\n`
var fs *client.Fsys
func prepare(t *testing.T){
    // checking for a running plumber instance
    var err error
    fs, err = client.MountService("plumb")
    if err == nil {
        t.Log("plumber_started_already")
    } else {
        // start plumber
        cmd := exec.Command("plumber")
        err = cmd.Run()
        if err != nil {
            t.Fatal(err)
        }
        t.Log("plumber_is_starting, wait a second")
        time.Sleep(time.Second)
    }
    fs, err = client.MountService("plumb")
    if err != nil {
        t.Fatal(err)
    }
    // setting a rule for the test
    f, err := fs.Open("rules", plan9.OWRITE)
    if err != nil {
        t.Fatal(err)
    }
    defer f.Close()
    _, err = f.Write([]byte(rule))
    if err != nil {
        t.Fatal(err)
    }
}
func compare(m1 *Message, m2 *Message) bool{
    if m1.Src != m2.Src ∨
        m1.Dst != m2.Dst ∨
        m1.Wdir != m2.Wdir ∨
        m1.Type != m2.Type ∨
        len(m1.Attr) != len(m2.Attr) {

```

```

    return false
}
for n, v := range m1.Attr {
    if m2.Attr[n] != v {
        return false
    }
}
return bytes.Compare(m1.Data, m2.Data) == 0
}
⟨Test routines 13⟩

```

4. At first let's describe *Message* structure. Actually it is almost the same like the original **Plan 9** C-struct. *Src* is a source of a message; *Dst* is a destination; *Wdir* is a working directory; *Type* is a type of the message, usually *text*; *Attr* is a slice of attributes of the message where an attribute is a pair of *name = value*; *Data* is a binary data of the message.

```

⟨Types 4⟩ ≡
    // Message describes a plumber message.
    Message struct{
        Src string
        Dst string
        Wdir string
        Type string
        Attr Attrs
        Data []byte
    }

```

See also sections 5 and 7.

This code is used in section 2.

5.

```

⟨Types 4⟩ +=
    // Attrs is a map of an attribute of a plumber message.
    Attrs map[string]string

```

6. A *Plumb* is a top-level structure. It contains a pointer to *os.File*, which is a port in **plumber's** file system. All fields of the *Plumb* are unexported.

```

⟨Imports 6⟩ ≡
    "9fans.net/go/plan9"
    "9fans.net/go/plan9/client"
    "os"

```

See also sections 10, 15, 17, 20, and 25.

This code is used in section 2.

```

7.  ⟨Types 4⟩ +=
    Plumb struct{
        f *client.Fid
        ⟨Other members of Plumb 30⟩
    }

```

**8. Open.** At first if *port* is not an absolute filename, a slash is added if necessary at the end of *port*. Then a file is opened with specified *omode*.

**9.** At first we have to mount **plumber** namespace

```
⟨ Variables 9 ⟩ ≡
    fsys * client.Fsys
    sp * Plumb
    rp * Plumb
```

This code is used in section 2.

**10.**

```
⟨ Imports 6 ⟩ +=
    "sync"
```

**11.**

```
⟨ Mount plumber namespace 11 ⟩ ≡
{
    var err error
    new(sync.Once).Do(func(){
        fsys, err = client.MountService("plumb")
    })
    if err != nil {
        return nil, err
    }
}
```

This code is used in section 12.

**12.**

```
// Open opens a specified port with a specified omode and returns *Plumb or error
func Open(port string, omode uint8) (*Plumb, error){
    ⟨ Mount plumber namespace 11 ⟩
    var p Plumb
    var err error
    if p.f, err = fsys.Open(port, omode); err != nil {
        return nil, err
    }
    return &p, nil
}
```

**13.** Let's test *Open* function.

⟨Test routines 13⟩ ≡

```
func TestOpen(t *testing.T){  
    prepare(t)  
    var err error  
    if sp, err = Open("send", plan9.OWRITE); err ≠ nil {  
        t.Fatal(err)  
    }  
    if rp, err = Open("goplumb", plan9.OREAD); err ≠ nil {  
        t.Fatal(err)  
    }  
}
```

See also sections 23, 27, 32, and 33.

This code is used in section 3.

**14. Send.** A *message* is packed and is written to the file.

```
// Send sends a message and returns error if any.
func (this *Plumb) Send(message *Message) error{
    if this == nil ∨ this.f == nil ∨ message == nil {
        return os.ErrInvalid
    }
    b := Pack(message)
    // a workaround: plumber can't receive a message with length more than 8192 - plan9.IOHDRSIZE
    for len(b)>0 {
        c := 8192 - plan9.IOHDRSIZE
        if len(b)<c {
            c = len(b)
        }
        c, err := this.f.Write(b[:c])
        if err != nil {
            return err
        }
        b = b[c:]
    }
    return nil
}
```



**15. Pack.** All the fields of a *message* are packed like a strings delimited by newlines.

⟨ Imports 6 ⟩ +≡  
"fmt"

**16.**

```
// Pack packs a message to []byte.
func Pack(message * Message) []byte{
    s := fmt.Sprintf("%s\n%s\n%s\n%s\n%s\n%d\n",
        message.Src, message.Dst,
        message.Wdir, message.Type,
        PackAttr(message.Attr),
        len(message.Data))
    b := append([]byte{}, []byte(s)...)
    return append(b, message.Data...)
}
```

**17. PackAttr.** Attributes *attr* are packed like pairs *Name* = *Value* delimited by spaces. *Value* can be quoted if it is necessary.

```
⟨Imports 6⟩ +=  
    "strings"
```

**18.**

```
// PackAttr packs attr to string. If an attribute value contains a white space,  
// a quote or an equal sign the value will be quoted.  
func PackAttr(attr Attrs) string{  
    var s string  
    first := true  
    for n, v := range attr {  
        if ¬first {  
            s += "␣"  
        }  
        first = false  
        if strings.ContainsAny(v, "␣'=\t") {  
            s += fmt.Sprintf("%s='%s'", n, strings.Replace(v, "'", "''", -1))  
        } else {  
            s += fmt.Sprintf("%s=%s", n, v)  
        }  
    }  
    return s  
}
```

**19. SendText.** A message is composed from  $Src = src$ ,  $Dst = dst$ ,  $Wdir = wdir$  and  $Type = text$

// *SendText* sends a text-only message; it assumes *Type* is text and *Attr* is empty.

// *SendText* returns **error** if any.

```
func (this *Plumb) SendText(src string, dst string, wdir string, data string) error{  
    m := &Message{  
        Src: src,  
        Dst: dst,  
        Wdir: wdir,  
        Type: "text",  
        Data: []byte(data)}  
    return this.Send(m)  
}
```

**20. Recv.** At most 8192 bytes are read for the first time. Then *UnpackPartial* is used to unpack a message. If the message too big *b* is reallocated for needed size, last part of the message is read and the message is unpacked.

```
⟨Imports 6⟩ +=
    "errors"
    "io"
```

**21.**

```
// Recv returns a pointer to a received message *Message or error.
func (this *Plumb) Recv() (*Message, error){
    if this == nil ∨ this.f == nil {
        return nil, os.ErrInvalid
    }
    b := make([]byte, 8192)
    n, err := this.f.Read(b)
    if err != nil {
        return nil, err
    }
    m, r := UnpackPartial(b[:n])
    if m != nil {
        return m, nil
    }
    if r == 0 {
        return nil, errors.New("buffer_too_small")
    }
    if r > len(b) - n {
        b1 := make([]byte, r + n)
        copy(b1, b)
        b = b1
    } else {
        b = b[:n + r]
    }
    _, err = io.ReadFull(this.f, b[n:])
    if err != nil {
        return nil, err
    }
    m, r = UnpackPartial(b)
    if m != nil {
        return m, nil
    }
    return nil, errors.New("buffer_too_small")
}
```

**22.** Let's test *Send* and *Recv* functions.

```
⟨Test specific imports 22⟩ ≡
    "errors"
```

This code is used in section 3.

23.  $\langle$ Test routines 13 $\rangle + \equiv$

```

func TestSendRecv(t *testing.T){
    var m Message
    m.Src = "Test"
    m.Dst = "goplumb"
    m.Wdir = "."
    m.Type = "text"
    m.Attr = make(Attr)
    m.Attr["attr1"] = "value1"
    m.Attr["attr2"] = "value2"
    m.Attr["attr3"] = "value_=_ '3\t"
    m.Data = []byte("1234567890")
    if err := sp.Send(&m); err ≠ nil {
        t.Fatal(err)
    }
    t.Logf("message_#v_has_been_sent\n", m)
    r, err := rp.Recv()
    if err ≠ nil {
        t.Fatal(err)
    }
    t.Logf("message_#v_has_been_received\n", *r)
    if ¬compare(r, &m) {
        t.Fatal(errors.New("messages_is_not_matched"))
    }
}

```

**24. Unpack.** *Unpack* just recalls *UnpackPartial* and ignores a rest of a message if the message is too big.

```
// Unpack return a pointer to an unpacked message *Message.  
func Unpack(b []byte) *Message{  
    m, _ := UnpackPartial(b)  
    return m  
}
```

**25. UnpackPartial.**

```

⟨Imports 6⟩ +=
  "bytes"
  "strconv"

```

**26.**

```

// UnpackPartial helps to unpack messages splited in peaces.
// The first call to UnpackPartial for a given message must be sufficient to unpack
// the header; subsequent calls permit unpacking messages with long data sections.
// For each call, b contains the complete message received so far.
// If the message is complete, a pointer to the resulting message m will be returned,
// and a number of remainings bytes r will be set to 0.
// Otherwise m will be nil and r will be set to the number of bytes
// remaining to be read for this message
// to be complete (recall that the byte count is in the header).
// Those bytes should be read by the caller, placed at location b[r:],
// and the message unpacked again.
// If an error is encountered, m will be nil and r will be zero.
func UnpackPartial(b []byte) (m *Message, r int){
  bb := bytes.Split(b, []byte{'\n'})
  if len(bb) < 6 {
    return nil, 0
  }
  m = &Message{Src: string(bb[0]), Dst: string(bb[1]), Wdir: string(bb[2]), Type:
string(bb[3]), Attr: UnpackAttr(string(bb[4]))}
  n, err := strconv.Atoi(string(bb[5]))
  if err != nil {
    return nil, 0
  }
  i := 0
  for j := 0; j < 6; j++ {
    i += len(bb[j]) + 1
  }
  if r = n - (len(b) - i); r != 0 {
    return nil, r
  }
  m.Data = make([]byte, n)
  copy(m.Data, b[i:i+n])
  return m, 0
}

```

**27.** Let's test *Send* and *Recv* functions with a big message.

⟨Test routines 13⟩ +≡

```

func TestSendRecvBigMessage(t *testing.T){
    var m Message
    m.Src = "Test"
    m.Dst = "goplumb"
    m.Wdir = "."
    m.Type = "text"
    m.Attr = make(Attr)
    m.Attr["attr1"] = "value1"
    m.Attr["attr2"] = "value2"
    m.Attr["attr3"] = "value_=_ '3\t"
    m.Data = make([]byte, 0, 9000)
    for i := 0; i < 900; i++ {
        m.Data = append(m.Data, []byte("1234567890")...)
    }
    if err := sp.Send(&m); err ≠ nil {
        t.Fatal(err)
    }
    t.Logf("message_#v_has_been_sent\n", m)
    r, err := rp.Recv()
    if err ≠ nil {
        t.Fatal(err)
    }
    t.Logf("message_#v_has_been_received\n", *r)
    if ¬compare(r, &m) {
        t.Fatal(errors.New("messages_is_not_matched"))
    }
}

```



**28. UnpackAttr.** *UnpackAttr* unpacks attributes from *s*, unquotes values if it is necessary.

```
// UnpackAttr unpack the attributes from s to Attrs
func UnpackAttr(s string) Attrs{
    attrs := make(Attrs)
    for i := 0; i < len(s); {
        var n, v string
        for ; i < len(s) ∧ s[i] ≠ '='; i++ {
            n += s[i:i + 1]
        }
        i++
        if i ≡ len(s) {
            break
        }
        if s[i] ≡ '\ ' {
            i++
            for ; i < len(s); i++ {
                if s[i] ≡ '\ ' {
                    if i + 1 ≡ len(s) {
                        break
                    }
                    if s[i + 1] ≠ '\ ' {
                        break
                    }
                }
                i++
            }
            v += s[i:i + 1]
        }
        i++
    } else {
        for ; i < len(s) ∧ s[i] ≠ '␣'; i++ {
            v += s[i:i + 1]
        }
    }
    i++
    attrs[n] = v
}
return attrs
}
```

**29. Close.** *Close* just closes an underlying *f*.

```
// Close closes the plumbing connection.  
func (this *Plumb) Close(){  
    if this ≠ nil ∧ this.f ≠ nil {  
        this.f.Close()  
        this.f = nil  
    }  
}
```

**30. MessageChannel.**

⟨ Other members of *Plumb* 30 ⟩ ≡

*ch* **chan** \* *Message*

This code is used in section 7.

**31.**

```
// MessageChannel returns a channel of *Message with a buffer size
// from which messages can be read or error.
// First call of MessageChannel starts a goroutine to read messages put them to the channel.
// Subsequent calls of EventChannel will return the same channel.
func (this * Plumb) MessageChannel(size int) (← chan * Message, error){
    if this ≡ nil ∨ this.f ≡ nil {
        return nil, os.ErrInvalid
    }
    if this.ch ≠ nil {
        return this.ch, nil
    }
    this.ch = make(chan * Message, size)
    go func(ch chan ← * Message){
        for m, err := this.Recv(); err ≡ nil; m, err = this.Recv() {
            ch ← m
        }
        close(ch)
    }(this.ch)
    return this.ch, nil
}
```

**32.** A test of *MessageChannel* function.

```

⟨Test routines 13⟩ +≡
func TestMessageChannel(t * testing.T){
    var m Message
    m.Src = "Test"
    m.Dst = "goplumb"
    m.Wdir = "."
    m.Type = "text"
    m.Attr = make(Attrs)
    m.Attr["attr1"] = "value1"
    m.Attr["attr2"] = "value2"
    m.Attr["attr3"] = "value_=_ '3\t"
    m.Data = []byte("1234567890")
    ch, err := rp.MessageChannel(0)
    if err ≠ nil {
        t.Fatal(err)
    }
    if err := sp.Send(&m); err ≠ nil {
        t.Fatal(err)
    }
    t.Logf("message_#v_has_been_sent\n", m)
    time.Sleep(time.Second)
    rm, ok :=← ch
    if ¬ok {
        t.Fatal(errors.New("messages_channel_is_closed"))
    }
    t.Logf("message_#v_has_been_received\n", *rm)
    if ¬compare(rm, &m) {
        t.Fatal(errors.New("messages_is_not_matched"))
    }
}

```

**33.** A test of *Close* function.

```

⟨Test routines 13⟩ +≡
func TestClose(t * testing.T){
    rp.Close()
    sp.Close()
}

```

**34. Index.**

*Atoi*: 26.  
*Attr*: 3, 4, 16, 19, 23, 26, 27, 32.  
*attr*: 17, 18.  
*attrs*: 28.  
*Attrs*: 4, 5, 18, 23, 27, 28, 32.  
*bb*: 26.  
*bytes*: 3, 25, 26.  
*b1*: 21.  
*ch*: 30, 31, 32.  
*client*: 3, 6, 7, 9, 11.  
*Close*: 3, 29, 33.  
*cmd*: 3.  
*Command*: 3.  
*compare*: 3, 23, 27, 32.  
*Compare*: 3.  
*ContainsAny*: 18.  
*Data*: 3, 4, 16, 19, 23, 26, 27, 32.  
*data*: 19.  
*Do*: 11.  
*dst*: 19.  
*Dst*: 3, 4, 16, 19, 23, 26, 27, 32.  
*err*: 3, 11, 12, 13, 14, 21, 23, 26, 27, 31, 32.  
*ErrInvalid*: 14, 21, 31.  
*errors*: 20, 22, 21, 23, 27, 32.  
*EventChannel*: 31.  
*exec*: 3.  
*Fatal*: 3, 13, 23, 27, 32.  
*Fid*: 7.  
*File*: 6.  
*first*: 18.  
*fmt*: 15, 16, 18.  
*fs*: 3.  
*Fsys*: 3, 9.  
*fsys*: 9, 11, 12.  
*goplumb*: 2, 3.  
*io*: 20, 21.  
*IOHDRSIZE*: 14.  
*Log*: 3.  
*Logf*: 23, 27, 32.  
*message*: 14, 15, 16.  
*Message*: 3, 4, 14, 16, 19, 21, 23, 24, 26, 27, 30, 31, 32.  
*MessageChannel*: 31, 32.  
*MountService*: 3, 11.  
*m1*: 3.  
*m2*: 3.  
*Name*: 17.  
*name*: 4.  
*New*: 21, 23, 27, 32.  
*ok*: 32.  
*omode*: 8, 12.  
*Once*: 11.  
*Open*: 3, 12, 13.  
*OREAD*: 13.  
*os*: 6, 14, 21, 31.  
*OWRITE*: 3, 13.  
*Pack*: 14, 16.  
*PackAttr*: 16, 18.  
*plan9*: 3, 6, 13, 14.  
*Plumb*: 6, 7, 9, 12, 14, 19, 21, 29, 31.  
*port*: 8, 12.  
*prepare*: 3, 13.  
*Read*: 21.  
*ReadFull*: 21.  
*Recv*: 21, 22, 23, 27, 31.  
*Replace*: 18.  
*rm*: 32.  
*rp*: 9, 13, 23, 27, 32, 33.  
*rule*: 3.  
*Run*: 3.  
*Second*: 3, 32.  
*Send*: 14, 19, 22, 23, 27, 32.  
*SendText*: 19.  
*size*: 31.  
*Sleep*: 3, 32.  
*sp*: 9, 13, 23, 27, 32, 33.  
*Split*: 26.  
*Sprintf*: 16, 18.  
*src*: 19.  
*Src*: 3, 4, 16, 19, 23, 26, 27, 32.  
*strconv*: 25, 26.  
*strings*: 17, 18.  
*sync*: 10, 11.  
*TestClose*: 33.  
*testing*: 3, 13, 23, 27, 32, 33.  
*TestMessageChannel*: 32.  
*TestOpen*: 13.  
*TestSendRecv*: 23.  
*TestSendRecvBigMessage*: 27.  
*text*: 4, 19.  
*this*: 14, 19, 21, 29, 31.  
*time*: 3, 32.  
*Type*: 3, 4, 16, 19, 23, 26, 27, 32.  
*Unpack*: 24.  
*UnpackAttr*: 26, 28.  
*UnpackPartial*: 20, 21, 24, 26.  
*Value*: 17.  
*value*: 4.  
*wdir*: 19.  
*Wdir*: 3, 4, 16, 19, 23, 26, 27, 32.  
*workaround for plumber*: 14.  
*Write*: 3, 14.

⟨ Imports [6](#), [10](#), [15](#), [17](#), [20](#), [25](#) ⟩ Used in section [2](#).  
⟨ Mount **plumber** namespace [11](#) ⟩ Used in section [12](#).  
⟨ Other members of *Plumb* [30](#) ⟩ Used in section [7](#).  
⟨ Test routines [13](#), [23](#), [27](#), [32](#), [33](#) ⟩ Used in section [3](#).  
⟨ Test specific imports [22](#) ⟩ Used in section [3](#).  
⟨ Types [4](#), [5](#), [7](#) ⟩ Used in section [2](#).  
⟨ Variables [9](#) ⟩ Used in section [2](#).  
⟨ `goplumb_test.go` [3](#) ⟩

# The goplumb package for manipulating plumb messages

(version 0.4.0)

	Section	Page
<b>Introduction</b> .....	<a href="#">1</a>	2
<b>Implementation</b> .....	<a href="#">2</a>	3
Open .....	<a href="#">8</a>	6
Send .....	<a href="#">14</a>	8
Pack .....	<a href="#">15</a>	9
PackAttr .....	<a href="#">17</a>	10
SendText .....	<a href="#">19</a>	11
Recv .....	<a href="#">20</a>	12
Unpack .....	<a href="#">24</a>	14
UnpackPartial .....	<a href="#">25</a>	15
UnpackAttr .....	<a href="#">28</a>	17
Close .....	<a href="#">29</a>	18
MessageChannel .....	<a href="#">30</a>	19
<b>Index</b> .....	<a href="#">34</a>	21

Copyright © 2013, 2020 Alexander Sychev. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- The name of author may not be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.